PROTOCOLO DE MONITORAMENTO E GERAÇÃO DE ESTÍMULOS PARA FPGAS EM AMBIENTE REMOTO

THIAGO CARNEIRO PITA

TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROTOCOLO DE MONITORAMENTO E GERAÇÃO DE ESTÍMULOS PARA FPGAS EM AMBIENTE REMOTO

THIAGO CARNEIRO PITA

Orientador: PROF. DR. DANIEL CHAVES CAFÉ, ENE/UNB

TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROTOCOLO DE MONITORAMENTO E GERAÇÃO DE ESTÍMULOS PARA FPGAS EM AMBIENTE REMOTO

THIAGO CARNEIRO PITA

TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Dr. Daniel Chaves Café, ENE/UnB
Orientador

Prof. Dr. Ricardo Zelenovsky, ENE/UnB
Examinador interno

Prof. Dr. Daniel Mauricio Muñoz Arboleda, FGA/UnB
Examinador interno

BRASÍLIA, 11 DE DEZEMBRO DE 2020.

FICHA CATALOGRÁFICA

THIAGO CARNEIRO PITA
Protocolo de Monitoramento e Geração de estímulos para FPGAs em ambiente remoto 2020xv, p., 201x297 mm
(ENE/FT/UnB, Bacharel, Engenharia Elétrica, 2020)
Trabalho de conclusão de curso de Graduação - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

THIAGO CARNEIRO PITA (2020) Protocolo de Monitoramento e Geração de estímulos para FPGAs em ambiente remoto. Trabalho de conclusão de curso de Graduação em Engenharia Elétrica, Publicação xxx/AAAA, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, p.

CESSÃO DE DIREITOS

AUTOR: Thiago Carneiro Pita
TÍTULO: Protocolo de Monitoramento e Geração de estímulos para FPGAs em ambiente remoto.
GRAU: Bacharel ANO: 2020

_____

Thiago Carneiro Pita
SQSW 101 Bloco F apt. 607

Agradecimentos

Agradeço ao meu orientador, professor Daniel Chaves Café, pela orientação e ajuda na realização deste trabalho, a todos os meus amigos e familiares que me ajudaram e me apoiaram durante a realização deste trabalho e à Universidade de Brasília pela formação profissional que me permitiu a realização deste trabalho.

Resumo

O desenvolvimento da eletrônica digital utiliza dispositivos de prototipação chamados FPGAs (Field programmable Gate Array) , mas este equipamento é de difícil acesso por pesquisadores e estudantes devido ao seu elevado custo. Estes equipamentos são acessíveis aos profissionais apenas nos laboratórios das instituições que as possuem. Essa baixa disponibilidade das FPGAs atrapalha o desenvolvimento dos projetos que precisam destes dispositivos e da formação de novos profissionais, prejudicando o desenvolvimento da eletrônica digital. A proposta para a solução deste problema é um sistema de acesso remoto de FPGAs. Essa solução é composta por três partes. Uma interface web para manipulação da FPGA, um sistema de geração de estímulos e captura de dados e um *software* de validação e programação da FPGA. A primeira parte foi desenvolvida por [Araújo 2019]. Este trabalho tem foco no sistema de geração de estímulos e captura de dados. O projeto será finalizado com um terceiro trabalho que irá focar num *software* de validação e programação da FPGA e integração dos diferentes componentes do sistema. A proposta desenvolvida permite o compartilhamento virtual dos equipamentos, facilitando a colaboração entre instituições. Este trabalho também propõe uma implementação inicial da solução utilizando um microcontrolador e um protocolo de comunicação simples para a manipulação do microcontrolador por um computador.

Palavras-Chave: FPGA, Laboratório Remoto, Ensino, Eletrônica Digital

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE TERMOS E SIGLAS

CDC	Communication Device Class
CLB	Configurable Logic Block
FPGA	Field programmable Gate Array
HID	Human Interface Device
IOB	Input/Output Blocks
JTAG	Joint Test Action Group
LED	Light-emitting diode
LUT	Look-up Table
RAM	Random Access Memory
ROM	Read-only Memory
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

Capítulo 1

Introdução

1.1 Problema

Hoje em dia praticamente todas as tecnologias de última geração são baseadas em dispositivos digitais e na capacidade de abstração criada por esses dispositivos. A interação aumenta a demanda por avanços na eletrônica digital e, consequentemente, dos equipamentos utilizados durante o aprendizado e desenvolvimento. Dentre estes dispositivos, as FPGAs (*Field Programmable Gate Array*) são importantes para realizar a prototipação dos projetos, mas são de difícil acesso. A principal dificuldade encontrada é relacionada com o custo do equipamento, que possui preço muito alto para ser adquirido por desenvolvedores e estudantes. Assim, os custos elevados para a obtenção das FPGAs as tornam acessíveis apenas para instituições de ensino e pesquisa e em baixas quantidades, que as disponibilizam para estudantes por meio da realização de cursos de capacitação. Assim, o acesso ao equipamento por estudantes só é possível durante a realização do curso e nos laboratório presenciais em horários determinados, devido a preocupação com segurança e integridade do equipamento que se danificado representa um prejuízo considerável à instituição e um atraso no desenvolvimento de projetos. A necessidade de um equipamento físico específico no desenvolvimento da eletrônica digital também dificulta a colaboração entre instituições de pesquisa, pois a ausência da FPGA em uma das instituições ou a utilização de modelos diferentes de FPGA limita as contribuições desta instituição de forma muito significativa.

O ano de 2020 foi marcado por um problema sanitário de altíssima importância, a pandemia criada pelo novo coronavírus, que teve como medida preventiva o distanciamento social e fechamento temporário de instituições. Estas medidas forçaram a adaptação dos modelos de trabalho e ensino no mundo, sendo transformados para modelos remotos. Consequentemente diminuindo ainda mais o acesso às FPGAs, que prejudicou tanto a formação de novos profissionais quanto o andamento adequado do desenvolvimento da área.

A adaptação dos modelos para sua realização remota se mostrou um desafio em situações que eram realizadas de forma totalmente presencial, como utilização de laboratórios,

restringindo ainda mais o acesso às FPGAs, que agora não poderiam ser utilizadas nem nos laboratórios da instituições que as possuem. As restrições de acesso a laboratórios podem surgir repentinamente e não são um problema exclusivo da eletrônica digital, mas de todas as áreas do conhecimento.

O difícil acesso às FPGAs por estudantes e desenvolvedores, devido aos elevados preços, é um grande problema no desenvolvimento de projetos e pesquisas na área, pois os trabalhos são restritos aos laboratórios em que as FPGAs estão e ao investimento necessário para treinar novos profissionais. Além disso, a limitação do uso do equipamento apenas de forma presencial torna o desenvolvimento do projeto muito susceptível a restrições de acesso aos espaços laboratoriais.

1.2 Proposta

Seria interessante permitir que estudantes utilizassem as FPGAs a qualquer momento, diminuindo as barreiras existentes no aprendizado da eletrônica digital, e que pesquisadores pudessem acessar os dispositivos de qualquer lugar. Possibilitando que o equipamento e o conhecimento fossem compartilhados de uma forma prática e acelerando o desenvolvimento de projetos que necessitam das FPGAs. Ambos os problemas, momento de acesso e local de utilização, já possuem propostas de solução como as realizadas por [Hashemian and Riddley 2007] ou [Indrusiak et al. 2007] que introduzem o conceito de laboratórios remotos e exemplificam por meio de implementações práticas diferentes. A ideia é montar em laboratório todos os equipamentos necessários para a utilização adequada da FPGA e conectar tudo a um servidor. Desta forma, qualquer pessoa com acesso ao computador será capaz de programar a FPGA e testar o projeto desenvolvido, sem a necessidade de contato direto com o equipamento, sendo que todo o processo será realizado apenas pelo computador. Dessa foma uma pessoa com acesso remoto ao computador poderia utilizar o dispositivo de qualquer lugar com acesso a internet, a qualquer momento do dia.

A figura 1.1 ilustra de forma abstrata a arquitetura proposta, que consiste de um dispositivo de geração de estímulos, um dispositivo de retorno, um servidor e um dispositivo em teste.

Na figura 1.1, os blocos de captura e geração de sinais representam os equipamentos conectados, como osciloscópios e geradores de função, e o dispositivo em teste que pode ser uma FPGA. A figura evidência o fluxo dos sinais no experimento e o objetivo de cada bloco do sistema, mas não define bem como cada parte deve ser implementada, consequentemente, cada implementação mistura e realiza os blocos de formas diferentes. Por exemplo [Gomes et al. 2009] utiliza um controlador PIC32 como servidor e 2 FPGAs para geração e recepção de estímulos com um circuito digital básico implementado em *protoboard* como dispositivo em teste. Outro exemplo é o desenvolvido por [Soares et al. 2011], que utiliza as ferramentas de depuração da fabricante, para manipular a memória da FPGA via JTAG

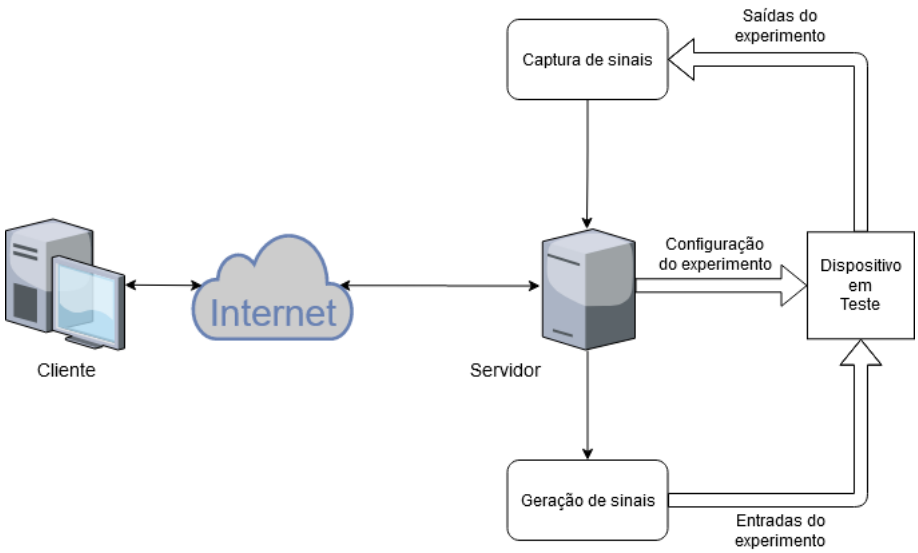

Figura 1.1: Diagrama abstrato do projeto

e gerar os sinais de estimulo internamente, para a captura da saída é utilizada as saídas da própria placa de desenvolvimento em conjunto com uma câmera para permitir a visualização remota do equipamento. Outros exemplos de foram implementadas por [Morgan et al. 2012] e [Rajasekhar et al. 2008] e serão discutidas mais a fundo na seção 2.1.

O diagrama da figura também pode ser utilizado em laboratórios de outras áreas do conhecimento, em que o maior desafio será na configuração remota do experimento, que na área de eletrônica digital é muito facilitada pelas FPGAs.

Este trabalho é uma continuação do trabalho realizado em [Araújo 2019], que focou seu desenvolvimento na interface de usuário, chamado de *frontend*, e no funcionamento do servidor, mas utilizou de forma bastante simples o microcontrolador e a FPGA. Basicamente utilizou estes dispositivos em seu trabalho apenas para a realização de testes das funcionalidades do sistema, como a gravação da FPGA e envio de bytes para o microcontrolador. O desenvolvimento deste trabalho é baseada na implementação proposta por [Araújo 2019] apresentada na figura 1.2, que utiliza um computador *desktop* com um servidor web, desenvolvido em *ruby*, um microcontrolador MSP430F5529 como dispositivo de geração e captura de sinais e uma FPGA como dispositivo em teste. Neste trabalho o foco será na melhoria do microcontrolador como dispositivo de geração de estímulos e captura de sinais, para permitir a realização de funções mais complexas durante o desenvolvimento e teste do *hardware* projetado na FPGA.

A figura 1.2 apresenta a mesma estrutura do digrama na figura 1.1, as diferenças são o colapso dos blocos de geração de sinais e captura de sinais no microcontrolador MSP430F5529 e na substituição do dispositivo em teste pela FPGA.

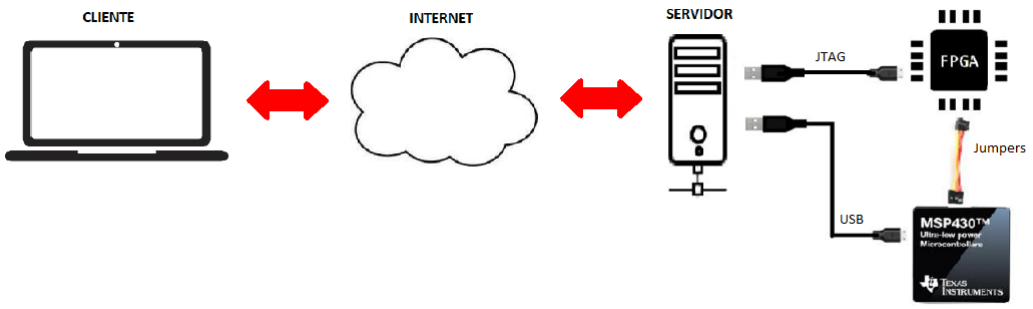

Figura 1.2: Proposta de implementação do projeto

1.3 Objetivos

Neste trabalho os objetivos principais são desenvolver um protocolo para a utilização remota dos periféricos de um microcontrolador e a implementação de um *firmware* para a manipulação e leitura de pinos utilizando o protocolo.

As especificações do protocolo são:

- Ser independente do controlador utilizado;
- Permitir o controle e gerenciamento de transações anteriores;
- Ter capacidade de expansão para atender requisitos futuros.

Para a conclusão deste trabalho será necessário implementar:

- Comunicação entre o servidor e o controlador;
- Manipulação dos pinos do controlador pelo servidor;
- Leitura dos pinos pelo controlador e envio ao servidor;
- Leitura periódica dos pinos pelo controlador e envio ao servidor.

É importante notar que as especificações de comunicação e dos testes devem ser independentes do microcontrolador, de forma que caso algum projeto necessite de um controlador diferente, com mais interfaces de comunicação ou mais veloz, esta alteração não deve afetar a implementação do servidor ou da comunicação, apenas a implementação do controlador.

Uma breve descrição das tecnologias utilizadas neste trabalho e dos trabalhos relacionados será apresentada no capítulo 2. As metodologias utilizadas para desenvolvimento e teste serão descritas no capítulo 3. No capítulo 4 está descrito o projeto do protocolo, assim como os pacotes desenvolvidos, a implementação do microcontrolador e uma descrição do código que roda no servidor utilizado para testar as implementações. Os resultados obtidos serão apresentados no capítulo 5. No capítulo 6 é descrito o resultado final do projeto e propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Trabalhos prévios

A grande maioria dos trabalhos sobre a implementação de laboratórios remotos são direcionados para o aprendizado, ou seja são focados em fornecer a melhor experiência possível para um usuário que está desenvolvendo suas habilidades e aplicando novos conceitos em seus experimentos. A maioria destas implementações seguem a arquitetura abstrata descrita em 1, basicamente a ideia é montar uma laboratório convencional, adicionar um servidor e conectar ao servidor todos os equipamentos do laboratório para que um usuário remoto possa realizar os experimentos apenas com acesso ao servidor. Nos laboratórios de engenharia os equipamentos normalmente são operados fisicamente, através de botões ou chaves e os resultados são obtidos por meio de estímulos visuais como LEDs e displays, ou seja, não podem ser diretamente conectadas a um servidor. Essa dificuldade é resolvida de forma distinta em cada implementação, por exemplo [Gomes et al. 2009] utiliza FPGAs para realizar a criação de estímulos e a obtenção de resultados. [Soares et al. 2011] utiliza uma câmera para permitir a leitura dos LEDs e *displays* da placa. Sendo que a geração de sinais é realizada pela manipulação da memória da FPGA pela interface JTAG. Para o funcionamento deste método, o projeto é modificado, alterando as entradas do experimento das chaves da placa para o endereço de memória que será manipulado. [Hashemian and Riddley 2007] utilizam dispositivos comuns em laboratórios como fontes de bancada e osciloscópios conectados ao servidor, sendo desnecessária a utilização de uma *webcam* para cada FPGA conectada ao sistema, mas exige um equipamento ainda mais caro e robusto. Com isso é possível a realização de testes mais complexos, como a verificação e validação de sinais sincronizados e rápidos que não podem ser observados em LEDs, por exemplo, comunicações seriais.

Na maioria dos casos o acesso ao servidor é realizado utilizando ferramentas de acesso remoto como *ssh* ou *Windows Remote Desktop*. As exceções [Araújo 2019] e [Soares et al. 2011], utilizam um servidor HTTP para gerenciamento das FPGAs em uso e manipulação dos equipamentos do laboratório. As funcionalidades presentes em cada implementação, também são bastante interessantes, por exemplo [Gomes et al. 2009] permite

ao estudante criar uma sequência de sinais a ser utilizada para testar o circuito e a observação dos diagramas temporais dos sinais, funções cruciais para a proposta do laboratório implementado, que difere da proposta deste trabalho. É interessante notar que a arquitetura apresentada nos estudos realizados pode ser expandida para outros laboratórios que possuam problemas de acesso semelhantes aos FPGAs. A proposta de [Hashemian and Riddley 2007] que utiliza gerador de funções, fonte de bancada e osciloscópio permite que este laboratório teste e valide sistemas com características analógicas, além das digitais, permitindo que sistemas muito mais complexos sejam testados, depurados e validados, sendo necessário apenas a presença de um técnico que realizará a montagem do sistema.

2.2 FPGA *(Field Programmable Gate Array)*

As FPGAs inicialmente eram compostas apenas por 3 unidades básicas as CLBs , IOBs e as matrizes de roteamento.

Os CLBs são unidades lógicas reconfiguráveis formadas por um conjunto de *look-up tables* (LUT) , implementações lógicas de tabelas verdade, um *full-adder* e um *flip-flop* tipo D, a configuração deste bloco é feita pela escolha dos sinais de saída da *look-up table* juntamente com os sinais de seleção dos multiplexadores que determinam quais LUTs serão utilizadas ou não em cada configuração. Os IOBs são os controladores dos sinais de entrada e saída das FPGAs, estes blocos configuram separadamente cada pino do FPGA como entrada ou saída do sistema, alguns IOBs possuem outros estados como coletor-aberto ou alta impedância. As matrizes de roteamento são bastante simples, consistem do cruzamento de 4 linhas de sinal com a interseção das linhas um conjunto de 6 transistores, um para cada opção de conexão, que realiza o chaveamento adequado do sinal de uma linha para outra.

Com essas três unidades cada CLB pode implementar uma função lógica simples arbitraria e ter os sinais de saída roteados de um pino de uma entrada até uma saída especifica passando por um conjunto de CLBs e portanto implementando funções lógicas complexas arbitrarias sobre cada entrada. Os IOBs permitem que o sistema implementado em FPGA possua a quantidade adequada de entradas e saídas, assim como a exposição de sinais internos para o escalonamento ou depuração do sistema.

A arquitetura descrita para a FPGA pode ser observada na figura 2.1. Nela as CLBs são os blocos marrons indicados por "*Logic Block*", as LUTs estão dentro das CLBs, os IOBs são os blocos verdes indicados por "*I/O Block*"e as matrizes de roteamento são representadas pela linha azul.

As FPGAs mais modernas possuem também processadores, normalmente ARM, memória além de outros periféricos mais complexos como interfaces seriais ou de depuração, tornando esses equipamentos muito mais robustos e permitindo ao desenvolvedor a criação de sistemas muito mais complexos com a utilização de um único chip e tornando o uso industrial deste equipamento muito mais viável.

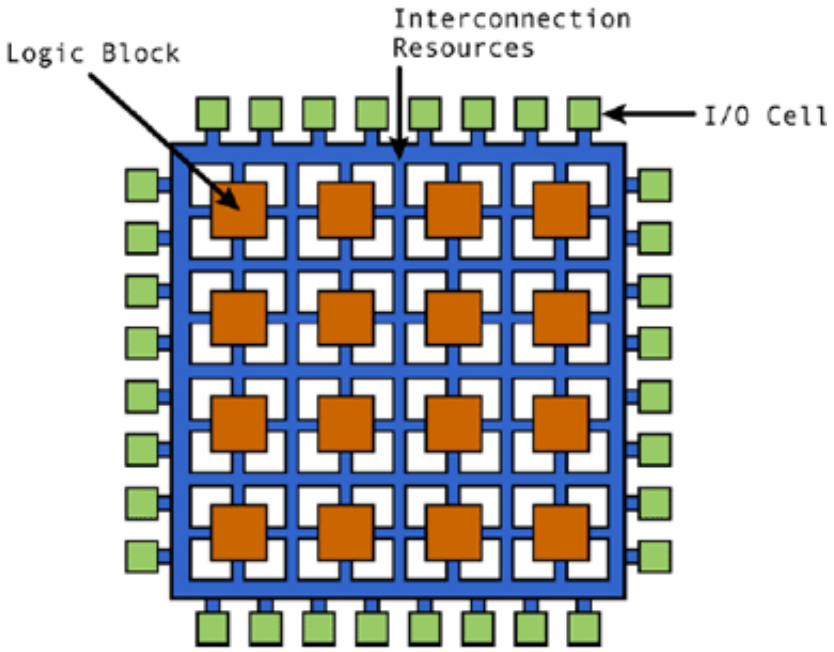

Figura 2.1: Arquitetura de uma FPGA

2.3 Microcontrolador

Os microcontroladores são dispositivos lógicos genéricos que possuem uma arquitetura lógica fixa, a arquitetura consiste basicamente em uma unidade de processamento, um barramento de periféricos e uma memória. É comum que a memória seja subdividida em seções uma para programa, uma para dados temporários. A memória também é dividida em tipos como *RAM* , *flash* ou *ROM* cada uma com objetivos e endereços diferentes. É comum que a *ROM* armazene o *bootloader* do microcontrolador, o programa seja guardado na memória *flash* onde pode ser alterado e a memória *RAM* normalmente é a memória de dados, que é perdida sempre que o microcontrolador é desligado. Alguns microcontroladores possuem outras seções de memória *flash* para dados sensíveis que não podem ser perdidos ao desligar o microcontrolador. A unidade de processamento é basicamente um grupo de registradores, uma ULA e um interpretador de instruções, os registradores indicam ao controlador onde na memória está a próxima instrução a ser executada. O interpretador busca a instrução, decodifica e indica aos registradores e à ULA qual operação deve ser realizada, para otimizar a performance do controlador é comum a utilização de um *pipeline* que consiste da paralelização dos processos de busca, decodificação e execução da instrução. Os periféricos do microcontrolador são pequenos sistemas que realizam alguma função além das capacidades da unidade de processamento, por exemplo *timers* ou comunicação serial. Na arquitetura de microcontrolador os periféricos utilizam o mesmo barramento de memória como forma de acesso.

A maioria das entradas e saídas de um sistema microcontrolado são feitas por meio dos periféricos do microcontrolador e diferentemente das FPGAs essa arquitetura não é reconfigurável, consequentemente cada pino possui apenas um conjunto de periféricos associados

que não podem ser re-roteados para outro pino de acordo com as necessidades do sistema. Este problema pode ser minimizado usando a técnica de bit-banging, que consiste na utilização dos pinos do microcontrolador para a emulação dos sinais gerados por algum periférico do microcontrolador. Um exemplo de *bit-banging* é a biblioteca *SoftwareSerial* disponível no pacote de desenvolvimento do arduino para os controladores ATMega328, esta biblioteca permite a utilização de uma interface serial UART em qualquer um dos pinos do microcontrolador, sua única limitação é o suporte do pino de recepção a interrupções, funcionalidade necessária para a captura assíncrona dos dados recebidos pelo pino.

Para este projeto foi utilizado o microcontrolador MSP430F5529 que é recomendado pela fabricante, *Texas Instruments*, para projetos de sensoriamento analógico e digital, *data logger* e conexão com *hosts* USB . Para a realização dos projetos, o microcontrolador possui 4 interfaces seriais, 1 interface USB, 4 *timers*, além de outros periféricos, e uma extensa documentação sobre sua utilização e a fabricante fornece vários exemplos que facilitam a realização de projetos. A qualidade da documentação fornecida pela fabricante do controlador faz com que o MSP430F5529 seja uma boa opção, não apenas para os projetos recomendados, mas também para o ensino de sistemas embarcados e sua utilização em projetos genéricos que necessitem de microcontroladores mais poderosos.

2.4 USB (Universal Serial Bus)

O USB é uma especificação industrial completa para as comunicações seriais entre dispositivos, assim como para a transferência de potência entre estes equipamentos de baixa tensão. Esta especificação determina os conectores, os cabos assim como as características destes cabos, os protocolos de comunicação, detecção de dispositivos, comportamento elétrico e alimentação de equipamentos.

A descrição da especificação no protocolo USB neste documento é válida até a versão 3.0 do protocolo.

Para o protocolo USB existem dois tipos de dispositivos, os *Hosts* e os *Devices* e cada dispositivo deve possuir um receptáculo fêmea de um dos tipos validos de conector, geralmente tipo A para o *Host* e qualquer outro tipo para o *Device*. Neste cabo existem ao menos 4 fios, dois com alimentação de 5V para alimentação do *Device* e um par diferencial trançado de 90 Ω para o tráfego de dados. A transmissão é feita de forma *half-duplex* e é baseada em um *clock* de 48 MHz para determinar a temporização dos bits e dos pacotes. O protocolo também especifica o método de detecção de *Device* pelo *Host*, que é basicamente a utilização de resistores de pull-up de 1.5 KΩ no *Device* e pull-down de 15 KΩ no pino de dados positivo para que quando não há *Device* conectado o nível lógico do pino seja zero e quando o Device é conectado o nível lógico é alterado para um.

A comunicação do protocolo USB é feita baseada em transferências e canais, chamados de *pipes*. Os *pipes* podem ser abertos e fechados livremente durante a comunicação, mas

são limitados em quantidade e configuração pelas capacidades do *Device*. Após aberto cada *pipe* deve ser configurado de acordo com o tipo de transferência que será realizado. A única exceção a essa regra é o *pipe* 0, que é aberto imediatamente após a conexão do *Device* e não pode ser fechado até a desconexão do dispositivo. Este *pipe* em especial também não precisa de configuração, pois é utilizado para realizar as transferências de controle entre os dispositivos, além das transferências de controle existem mais 3 tipos de transferência no protocolo USB, cada uma com sua particularidades de funcionalidade e usabilidade.

2.4.1 Transferência de controle

Este tipo de transferência é feita em rajadas curtas e não periódicas, o tamanho máximo do pacote vai até 64 bytes a depender das capacidades do *Device*, o *pipe* configurado para este tipo de transferência é bidirecional e as mensagens sempre tratam sobre a configuração do *Device* ou do seu *Status* ou então sobre informações de controle. Este método devido ao trafego de informações sensíveis ao funcionamento adequado de ambos os equipamentos exige a presença de técnicas robustas de garantia de entrega dos dados, como retransmissão e detecção de erros nos pacote.

2.4.2 Transferência Isócrona

Este tipo de transferência é ideal para transferência de informações que são *time-dependant*, as transmissões são feitas periodicamente ou continuamente entre *Device* e *Host*, mas cada *pipe* é unidirecional e portanto se faz necessário a utilização de 2 *pipes* para que a comunicação possa ser bidirecional, as principais características deste tipo de transferência são a latência limitada e a taxa de transmissão fixa.

2.4.3 Transferência por interrupção

Este tipo de transferência é realizada de forma periódica e cada *pipe* é unidirecional, assim como a transferência isócrona, mas com intervalos maiores e com processos de detecção de erros e retransmissão de dados caso seja necessário, além disso o tamanho máximo dos pacotes pode ser de 8, 64 ou 1024 bytes a depender das capacidades do *Device*. Este método é o ideal para *Devices* que realizem transmissões de poucos dados esporadicamente e com latência limitada.

2.4.4 Transferência em massa

Este tipo de transmissão é realizada em rajadas e com grandes pacotes de acordo com a disponibilidade de banda pelos outros *pipes* neste método o *pipe* também é unidirecional e possui as funcionalidades de detecção de erro e retransmissão, este método possui garantia

de transferência, sem nenhuma garantia de banda ou latência sendo ideal para a transmissão de dados que não sejam sensíveis ao tempo e relativamente grandes, como arquivos para armazenamento.

2.4.5 Classes de Dispositivos

Nas especificações do protocolo USB os dispositivos são organizados em classes de acordo com as suas necessidades de transmissão de dados, suas funcionalidades e seu comportamento, as classes de dispositivos também são uteis para permitir ao computador decidir a melhor forma de interagir com o dispositivo. Um exemplo de classes definidas no protocolo são os *Human Interface Device* (HID) que é utilizada para dispositivos de interação entre seres humanos e o computador como *mouses* e teclados. Na definição dessa classe as transferências utilizadas são apenas as de controle e de interrupção. Outros exemplos de classes são a *Mass Storage Class*, usada por *pendrives* e *HDs* externos e transfere os dados principalmente por transferências em massa, e a *Communication Device Class* (CDC) que é utilizada como classe genérica para os dispositivos que não se classificam em outras classes como conversores serial-USB de microcontroladores, tem acesso a todos os tipos transferência e pode apresentar praticamente qualquer comportamento, por exemplo emulando comunicações seriais.

2.5 Testes automatizados

O processo de desenvolvimento, tanto para *software* quanto para *hardwares*, consiste de um conjunto de etapas, estas são:

- Planejamento;
- Implementação;
- Teste;
- Implantação.

Na etapa de planejamento são determinados, baseados no problema que será resolvido, todos os requisitos e características do sistema a ser desenvolvido, além disso o sistema é diagramado de forma a satisfazer os tais requisitos e são geradas as tarefas e tempo de realização de cada tarefa para que o sistema possa ser concluído.

A etapa de implementação é feita quase que em paralelo com a etapa de teste. Nesta etapa são realizadas as tarefas definidas no planejamento, o sistema é dividido em módulos que podem ser desenvolvidos e testados separadamente, são criados os testes específicos de cada módulo e o sistema começa a ser construído. Quando um módulo é terminado, ele é

adicionado ao sistema e é executada a regressão dos testes, todos os módulos do sistema são testados novamente para garantir que a adição do novo módulo não interferiu no correto funcionamento dos outros módulos. A realização do mesmo teste repetidamente é uma atividade cansativa e ineficiente, pois para cada alteração é necessário a realização de todos os testes do sistema, mas a solução deste problema é simples, desenvolver os testes na forma de um programa que pode ser executado por um computador.

A geração dos testes automatizados é baseada no conhecimento do desenvolvedor do módulo e da sua capacidade de gerar uma quantidade arbitrária de entradas com suas respectivas saídas para que, ao alimentar uma implementação correta do sistema com estímulos predefinidos, as saídas esperadas e a real sejam sempre iguais. Uma máquina pode realizar este processo de comparação automaticamente e avisar ao desenvolvedor quando algum teste falha. No caso de desenvolvimento de *hardware* o comportamento mais próximo dos testes automatizados que pode ser feito é a realização de simulações e comparação entre o projeto e seu *Golden Model*, uma implementação que possui a mesmo comportamento esperado do projeto, mas muitas vezes as simulações não são perfeitas e não consideram alguns aspectos do sistema real que podem gerar falhas e portanto criando a necessidade de um desenvolvedor testar manualmente cada parte da implementação mesmo após o sucesso dos testes simulados.

A etapa de implantação consiste na utilização, continua melhoria e ajuste do sistema. No caso de *softwares* é mantida a regressão dos testes do sistema, com o objetivo de garantir o adequado funcionamento do sistema após a realização de cada ajuste necessário. No caso de *hardwares* os projetos desenvolvidos são incluídos às simulações e são utilizadas para comparar as novas arquiteturas e versões do *hardware* com o mesmo nível de automação que os testes de *software*, mas com o mesmo problema citado anteriormente.

Capítulo 3

Metodologia

O desenvolvimento do projeto será baseado em um exemplo fornecido pela empresa Texas Instruments, fornecedora do microcontrolador utilizado no projeto, com foco nos objetivos definidos no capítulo 1. O exemplo utilizado é o *C3_EchoToHost* que utiliza a interface USB do microcontrolador em modo CDC para emular uma interface serial e retorna ao computador os bytes enviados ao microcontrolador. Este modo foi escolhido pela liberdade de acesso aos dados e compatibilidade com outros microcontroladores que não possuam interface USB, pois este é o mesmo modo que um conversor USB-Serial funciona, esta compatibilidade é uma característica interessante para o sistema, mencionada no capítulo 1.

O desenvolvimento dos objetivos do projeto foram realizados em paralelo com o desenvolvimento de um *software* de teste para ser executado pelo computador. A cada objetivo atingido no desenvolvimento do *firmware* um novo teste foi implementado no *software* para validar a nova funcionalidade. Uma funcionalidade foi considerada concluída apenas quando aprovada em todos os testes e teve sua execução verificada utilizando um analisador lógico ou os componentes de teste da placa. Seguindo a ideia de regressão, para garantir o completo funcionamento do *firmware*, os testes foram realizados sempre em conjunto com a validação das funcionalidades anteriores, assim garantindo que a implementação de uma nova funcionalidade não criou problemas em partes que já haviam sido finalizadas.

O *Software* desenvolvidos consiste basicamente do envio de comandos periodicamente para o microcontrolador e monitoramento da resposta, assim a validação dos testes foi realizada pela leitura dos dados pelo desenvolvedor junto a manipulação dos pinos do microcontrolador. Durante a realização dos testes, além das chaves e LEDs da placa, também foi utilizado para a validação das funcionalidade implementadas, alguns pinos em curto circuito, possibilitando a validação do comportamento apenas pela leitura dos dados transmitidos entre o microcontrolador e o computador.

Os testes desenvolvidos no projeto consistem no envio dos comandos implementados no projeto, realizado na forma de um *software* para ser utilizado como base de um trabalho futuro de integração entre este trabalho e o trabalho desenvolvido por [Araújo 2019]. Com

o objetivo de simplificar esta integração o software foi desenvolvido em *Ruby*, a mesma linguagem utilizada no servidor desenvolvido por [Araújo 2019].

A sequência de comandos utilizada consiste na configuração de alguns pinos como saída, especificamente os pinos 2.1, 1.1 e 6.5, e alguns pinos como entrada, especificamente os pinos 1.0, 4.7 e 6.2, em seguida é configurado o monitoramento do pino 1.1 e 6.2 com períodos de 0.5 e 1 segundos respectivamente. Finalmente é iniciado um *loop* em que o pino 2.1 é lido periodicamente e os outros pinos configurados como saída são manipulados com uma sequência pseudo-aleatória gerada e conhecida para que o resultado possa ser validado posteriormente, assim o funcionamento do sistema pode ser observado pelos LEDs conectados nos pinos 2.1 e 1.1, e pelo monitoramento do pino 6.2 através de um *jumper* conectando os pinos 6.5 e 6.2 ou utilizando um analisador lógico para a leitura das saídas.

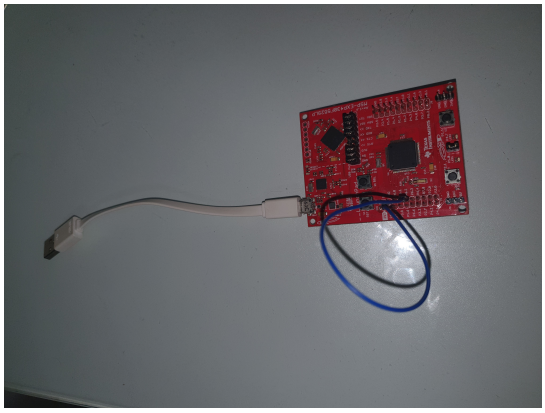

Figura 3.1: Montagem para a realização dos testes

Na figura 3.1 podemos ver a montagem necessária para a realização do experimento, que consiste da placa MSP430F5529LP com um curto circuito entre os pinos 6.2 e 6.5. O experimento não utilizou nenhuma FPGA durante os testes devido a falta de acesso ao equipamento e por ser desnecessário a sua utilização, pois o próprio microcontrolador pode gerar os sinais para validação do sistema.

Os testes são executados com o comando 'ruby main.rb' no diretório em que se localiza o *software* desenvolvido, após a conexão do microcontrolador ao computador. A validação dos testes não pode ser feita de forma automática devido a utilização dos LEDs e chaves no teste, mas pode ser facilmente automatizado para o *loopback* mostrado na figura 3.1, comparando o sinal gerado ao sinal lido nos pinos.

Além de verificar o funcionamento do sistema também é desejável obter alguns parâmetros do sistema, sendo eles:

- Latência de execução do comando;
- Taxa máxima de amostragem.

Para a obtenção deste parâmetros, serão utilizados alguns pinos como sinalizador de execução, sendo manipulados de forma síncrona aos eventos importantes da operação do micro-

controlador e permitindo a visualização do tempo de execução de cada tarefa do controlador. Para facilitar a realização das medidas, o sinal que representa a amostragem terá o estado alterado sempre que a amostragem for realizada, todos os outros sinais serão mantidos em nível lógico alto enquanto a sua tarefa esta sendo realizada. Neste experimento serão utilizados os pinos P2.0, P2.2, P2.4, P1.5, P1.4, P1.3 e P1.2 que serão configurados como saída do controlador independente dos comandos enviados pelo servidor. A manipulação destes pinos será feita nas tarefas de:

- Recepção de dados do servidor;
- Processamento do cabeçalho do protocolo;
- Execução do comando;
- Geração do pacote e envio ao servidor;
- Amostragem do pino monitorado.

A visualização dos sinais será feita utilizando um analisador lógico. Para a obtenção da taxa máxima de amostragem o *software* de teste será executado várias vezes variando o período de amostragem inicial até seja obtida uma onda quadrada estável no sinal de visualização. A latência de execução de comandos será calculada pela soma dos atrasos de recepção, processamento e execução. O valor desejado para a latência de execução de comandos é de menos de 1 mili-segundo. Para a taxa máxima de amostragem é esperado que o limitante seja o processador ou o *clock* utilizado no *timer*, assim o valor estimado é entre 30 e 300 micro-segundos.

Para a realização deste experimento foi realizada uma nova montagem, apresentada na figura 3.2, nela podemos ver o microcontrolador com o analisador lógico conectado aos pinos P2.0, P2.2, P2.4, P1.5, P1.4, P1.3 e P1.2, ambos conectados ao computador para o envio dos comandos pelo software de teste.

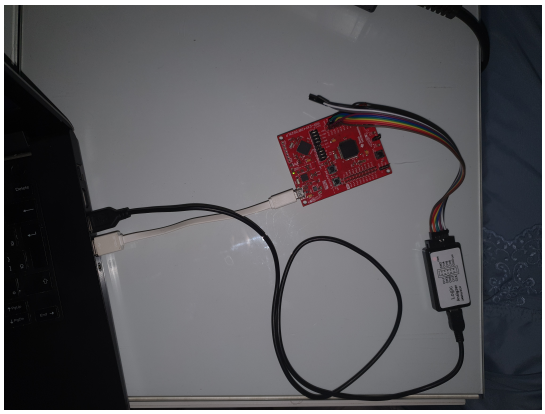

Figura 3.2: Montagem para a caracterização do sistema

O *software* de teste foi implementado em um computador com sistema operacional Linux Ubuntu 20.04. Foram observados problemas de compatibilidade entre o módulo serial do

ruby e o sistema operacional Windows. As incompatibilidades encontradas não serão um problema, pois o servidor desenvolvido por [Araújo 2019] foi testado em ambiente Linux e os computadores que serão utilizados como servidor possuem este sistema operacional.

Capítulo 4

Desenvolvimento

4.1 Protocolo

Um requisito do projeto foi a criação de um protocolo que permitisse o acesso e controle dos periféricos do microcontrolador, possibilitando a geração e captura de sinais remotamente. O protocolo foi desenvolvido sem levar em consideração o microcontrolador a ser utilizado na implementação final do projeto, para que caso haja a necessidade de alteração do microcontrolador posteriormente, isso possa ser feito sem muito esforço. O protocolo foi dividido em duas partes: o cabeçalho e o *payload*. O cabeçalho que possui informações de controle do protocolo e informações de operações que sejam independentes do microcontrolador, como configuração de pinos e *timers*. O *payload* que possui todas as informações necessárias para a execução do comando, quais pinos ou *timers* devem ser configurados e como cada objeto deve ser configurado. É importante perceber que toda a informação dependente do microcontrolador está exclusivamente no *payload* e portanto é a única parte do protocolo que deve ser ajustada caso o microcontrolador seja trocado.

O cabeçalho do protocolo é mostrado na figura 4.1 e possui 3 objetivos. 1) informar ao controlador qual ação será realizada; 2) indicar quais informações esperar no *payload* e 3) controlar e gerenciar as transações anteriores do protocolo. Para alcançar o objetivo 1 foram utilizados os campos de *command* e *type*, cada campo com 1 *nibble*, onde o campo de *command* informa a ação a ser realizada, como estimular, ler ou configurar o periférico, e o campo de tipo informa qual periférico a operação será executada. O objetivo 2 é alcançado pelo conjunto dos campos utilizados para o objetivo 1 em conjunto com o campo de *length* que indica o tamanho do *payload* que deverá ser processado. O objetivo 3 é alcançado com o campo *label* e o campo *length*. O campo *label* possui um identificador que pode ser utilizado para referenciar o pacote no futuro e o campo *length* permite uma validação simples do *payload* recebido. Ambos os campos *label* e *length* podem assumir qualquer valor. O tamanho do *payload* é limitado em 255 bytes devido ao tamanho do campo *length* no cabeçalho ser de 8 *bits*. O cabeçalho traz uma região reservada que foi pensada para permitir a expansão do protocolo caso seja necessária futuramente, por exemplo permitindo

a criação de *flags* que indiquem modificações do cabeçalho ou *checksum* para melhorar a validação dos dados.

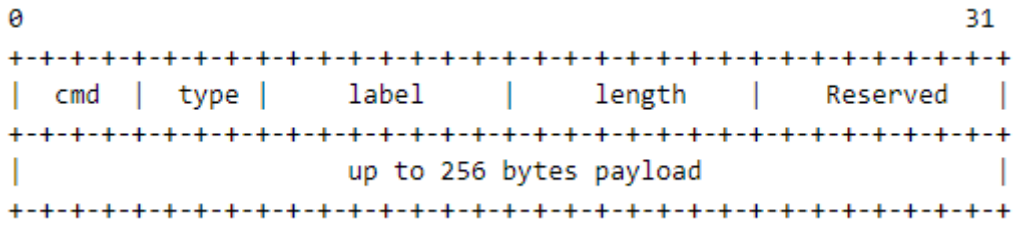

Figura 4.1: Estrutura do protocolo desenvolvido

O uso dos campos do cabeçalho são resumidos

Campo	**Tamanho**	**Descrição**
Command	4bits	Indicação da operação a ser realizada pelo controlador. 1h = Configuração 2h = Estímulo 3h = Leitura 7h = Monitoramento
Type	4 bits	Indicação do periférico em que o comando será executado. 2h = Pino
Label	8 bits	Identificação do pacote no contexto do protocolo. Pode assumir qualquer valor de 0 até 255.
Length	8 bits	Indicação da quantidade de bytes no *payload*. Pode assumir qualquer valor de 0 até 255.

Tabela 4.1: Campos do cabeçalho de envio e possíveis valores

Campo	**Tamanho**	**Descrição**
Command	4 bits	Indicação do motivo de envio da mensagem pelo controlador. 2h = Notificação 4h = Execução de estímulo 5h = Erro
Type	4 bits	Complemento do motivo de envio da resposta. 2h = Pino 5h = Configuração 6h = Monitoramento
Label	8 bits	Identificação do pacote no contexto do protocolo. Pode assumir qualquer valor de 0 até 255.
Length	8 bits	Indicação da quantidade de bytes no *payload*. Pode assumir qualquer valor de 0 até 255.

Tabela 4.2: Campos do cabeçalho de resposta e possíveis valores

Devido ao foco dado aos objetivos do projeto, definidos no capitulo 1, foram desenvolvidas apenas algumas operações necessárias para realizar cada objetivo, sendo elas:

- Operação de configuração;
- Operação de geração de estímulo;
- Operação de leitura;
- Operação de monitoramento de pino.

Cada operação pode ser ampliada pelo uso dos tipos, mas como os objetivos deste projeto são todos relacionados à manipulação dos pinos do microcontrolador, então todas as operações são realizadas utilizando o mesmo tipo.

Para a realização da configuração dos pinos são necessários como parâmetros o pino a ser configurado e o modo de configuração (*input*, *output*, *input* com resistor de *pulldown*, etc), sendo assim o *payload* deste comando é uma sequencia de pares [pino; modo], com cada parâmetro representado por 1 *byte*. O comando de configuração é representado pelo *nibble* '1'.

Na figura 4.2 é apresentada, como exemplo, uma sequência de *bytes* que corresponde ao comando de configuração dos pinos, sendo 1 configuração de saída e 2 de entrada. O pacote traduzido como configuração de pinos, byte 0x12, identificado como 0x00 e possui 6 parâmetros no *payload*, pino 1.0 como saída, par [0x10; 0x01], pino 6.2 como entrada sem resistor, par [0x62; 0x02] e pino 2.1 como entrada com resistor de *pullup*, par [0x21; 0x04].

[0x12 0x00 0x06 0x00 0x10 0x01 0x62 0x02 0x21 0x04]

Figura 4.2: Exemplo de pacote de configuração

A operação de geração de estímulo para a manipulação de pinos segue o mesmo princípio de funcionamento e portanto possui o mesmo formato do *payload*, este pacote é exemplificado na figura 4.3. O exemplo da figura possui comando de geração de estímulo de pino, representado pelo byte 0x22, realiza o *toggle* do pino 1.0, representado pelo par [0x10; 0x03] e seta o nível lógico do pino 4.7 em baixo, par [0x47; 0x02].

[0x22 0x00 0x04 0x00 0x10 0x03 0x47 0x02]

Figura 4.3: Exemplo de pacote de estímulo

A operação de leitura é mais simples e consequentemente necessita de menos parâmetros para sua execução, assim o seu *payload* consiste exclusivamente na lista de pinos a serem lidos pelo microcontrolador. O pacote deste comando é exemplificado na figura 4.4 e significa leitura de pinos, byte 0x32, do pino 2.1, byte 0x21.

A resposta enviada pelo microcontrolador tem um comportamento um pouco diferente. O retorno do microcontrolador possui 3 operações e 3 tipos, que foram utilizadas em conjunto para representar as respostas necessárias. As operações criadas foram:

[0x32 0x00 0x01 0x00 0x21]

Figura 4.4: Exemplo de pacote de leitura

- Operação de notificação;
- Operação de execução;
- Operação de erro.

Os tipos utilizados na criação das respostas foram:

- Tipo pino;
- Tipo configuração;
- Tipo monitoramento.

A resposta de erro consiste apenas no cabeçalho com a operação de erro e o resto do cabeçalho preenchido com zeros. Esta resposta significa que a combinação de operação e tipo recebido pelo microcontrolador não foi implementada. A resposta de notificação é um tipo de resposta genérica que é enviada após o recebimento de um comando válido do computador. Esta operação possui um *payload* que indica quais conjuntos de parâmetros recebidos foram validados ou os resultados obtidos, as respostas de notificação estão exemplificadas nas figuras 4.5, 4.6 e 4.7. A resposta de execução indica a execução de um estímulo pelo microcontrolador. O *payload* desta resposta é a mesma sequência de pinos enviada ao controlador seguida de um byte indicando se a manipulação recebida foi reconhecida e executada, esta resposta é exemplificada na figura 4.8.

[0x25 0x00 0x03 0x00 0x01 0x00 0x01]

Figura 4.5: Exemplo de notificação de configuração com falha na execução do segundo par

[0x26 0x00 0x02 0x00 0x011 0x00]

Figura 4.6: Exemplo de notificação de monitoramento com valor do pino 1.1

[0x22 0x00 0x02 0x00 0x21 0x01]

Figura 4.7: Exemplo notificação de leitura do pino 2.1 com o valor obtido

[0x42 0x00 0x02 0x00 0x01 0x01]

Figura 4.8: Exemplo de resposta de execução, com sucesso na escrita de dois pares

Os *payloads* de todos os pacotes utilizados serão mostrados no capítulo 5, na apresentação dos resultados obtidos dos testes.

4.2 Microcontrolador

O *firmware* desenvolvido foi baseado no exemplo fornecido pela fabricante do microcontrolador *C3_EchoToHost*, que utiliza o pacote *MSP430Ware* também disponibilizado pela *Texas Instruments*. Este exemplo inicialmente realiza uma etapa de *setup* e executa um *loop* onde recebe informações pela interface USB do microcontrolador em plano de fundo, controlado por interrupções, enquanto espera em modo de baixo consumo. Quando os dados são recebidos o microcontrolador desperta e adiciona os dados na fila de envio. O *firmware* desenvolvido utiliza a mesma estrutura, processando e executando o comando recebido em primeiro plano, fluxo normal do programa sem interrupções, enquanto a comunicação USB e monitoramento são realizados em plano de fundo. A etapa de *setup* consiste na configuração prévia de alguns periféricos que serão utilizados, como os *timers* e GPIOs. Nesta etapa também são configurados os *clocks* do microcontrolador. Sendo o *Master clock* com 25 MHz, o *Subsystem Master clock* com 25 MHZ, o *Phase locked loop* com 48 MHz e o *Auxiliary clock* com 32 kHz. O modo de baixo consumo foi retirado do *firmware* para evitar interferências e *delays* com a utilização dos periféricos.

O *firmware* desenvolvido para o MSP430 permite a utilização de todos os GPIOs do microcontrolador de forma remota com as três opções básicas de manipulação implementadas, sendo elas *High*, *Low* e *toggle*. Uma limitação é em relação ao monitoramento dos pinos, que esta limitada a um pino por *timer* do microcontrolador. Devido a esta limitação o *payload* do comando de monitoramento possui três parâmetros, pino a ser monitorado, *timer* que deve ser utilizado para o monitoramento e o tempo de amostragem para a observação adequada do sinal. A função de monitoramento inicialmente configura a interrupção do *timer* com o tempo de amostragem desejado e em cada interrupção gerada pelo *timer* o pino é lido e o resultado é enviado ao computador. O código desenvolvido também não tornou acessível ao usuário a utilização dos outros periféricos do controlador, como interfaces seriais ou geração de PWM. Esta limitação é aceitável, pois estas funcionalidades não são necessárias para o funcionamento básico do projeto, pois podem ser emuladas pela manipulação adequada dos GPIOs. O aperfeiçoamento deste projeto utilizando os periféricos do microcontrolador para otimizar a geração de sinais complexos, como comunicação serial ou PWM, é uma boa opção de trabalho futuro, junto da melhoria do uso do *timer* para monitoramento dos pinos e a utilização do *Subsystem Master clock* para aumentar a taxa máxima de amostragem do sistema.

As limitação mencionadas são referentes ao *firmware* desenvolvido e outras implementações ou versões futuras deste trabalho podem não ter essas limitações.

Capítulo 5

Resultados

A montagem do microcontrolador para a realização dos testes, conforme descrito e ilustrado no capítulo 3, consiste simplesmente no microcontrolador com um *jumper* conectando os pinos 6.2 e 6.5, de forma que o sinal gerado no pino 6.5 possa ser capturado pelo pino 6.2. Após a montagem, o microcontrolador é conectado ao computador e o *software* de teste é executado, os resultados obtidos são apresentados na figura a 5.1.

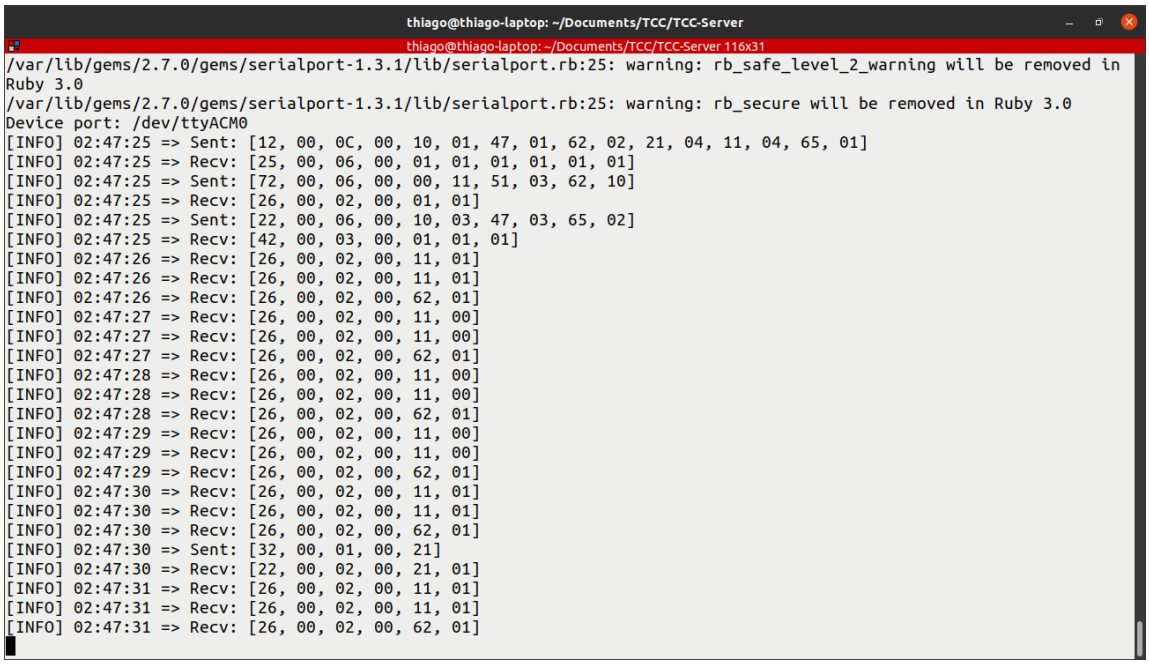

Figura 5.1: Resultado da execução dos testes

Na figura 5.1 podemos observar os comandos enviados ao controlador e a resposta a cada comando. As duas primeiras linhas representam a configuração dos pinos do controlador, pacotes iniciados com 0x12 e 0x25 respectivamente, como descrito no capítulo 4. Logo em seguida é configurado o monitoramento dos pinos 1.1 e 6.2, pacote inciado com 0x72, utilizando os *timers* A0 e B0 do microcontrolador, com os tempos de amostragem de 0.5 e 1 segundos, seguido da manipulação dos pinos de saída, iniciado com 0x22. O tempo de amostragem foi validado utilizando o horário do relógio disponibilizado junto a mensagem.

Utilizando essa ferramenta podemos ver que a notificação do pino 1.1 é recebida 2 vezes por segundo e a notificação do pino 6.2 é recebida apenas uma vez. Podemos ver na figura vários pacotes de monitoramento, iniciado com 0x26, e um pacote de pedido, iniciado com 0x32, seguido da resposta com o valor do pino. Pelas mensagens de monitoramento do pino 1.1, representado pelo byte 0x11, podemos ver que a chave da placa foi pressionada, causando a leitura do pino em nível baixo durante um período e depois foi liberada, retornando a leitura para o valor alto. O monitoramento também pode ser observado na figura 5.2.

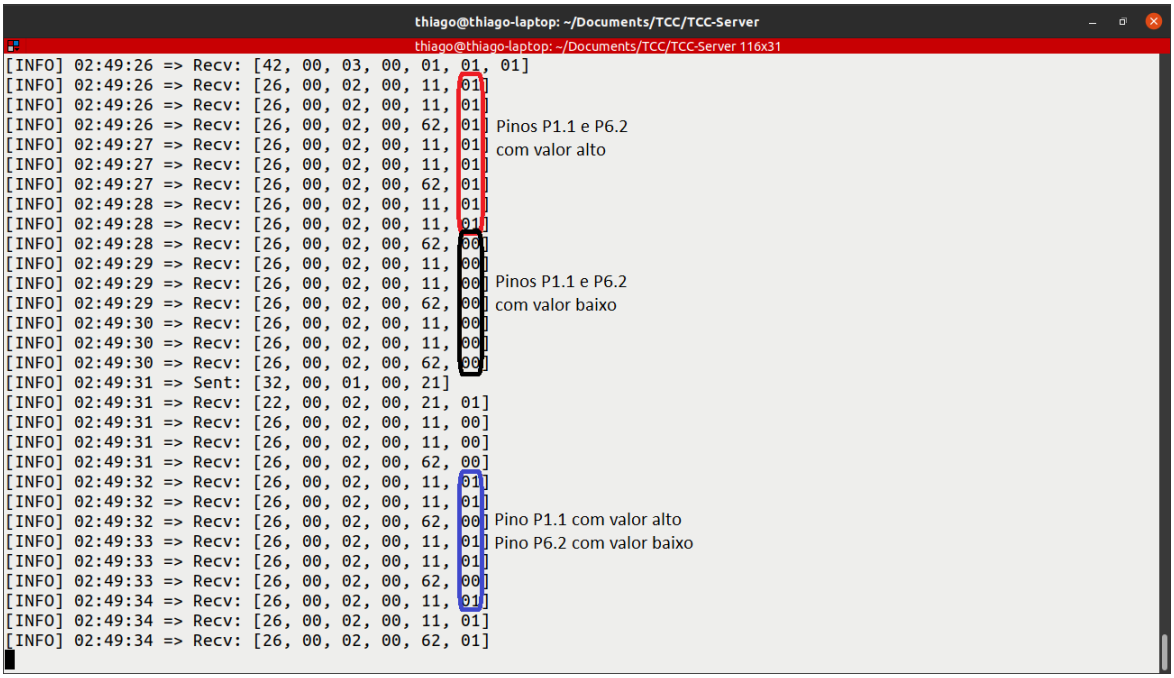

Figura 5.2: Exemplo de monitoramento

Nesta figura podemos observar o funcionamento do monitoramento de pinos, pois é possível observar o comportamento do sinal presente no pino que se manteve em nível lógico alto, foi alterado para nível baixo por 3 segundos e depois foi novamente alterado para alto.

Os resultados dos testes de geração de estímulos foram obtidas com um analisador lógico para facilitar a visualização do sinal. Para isso o *loop* entre os pinos 6.2 e 6.5 foi desfeito e um analisador logico foi conectado ao pino configurado como saída. O resultado obtido da realização deste teste é um sinal gerado aleatoriamente, com uma possível alteração de estado a cada 10 segundos, este sinal pode ser observado nas figuras 5.3 e 5.4. Sendo na figura 5.3 o sinal amostrado por 5 minutos e na figura 5.4 o mesmo sinal com um *zoom* para visualização do tempo entre as mudanças de estado.

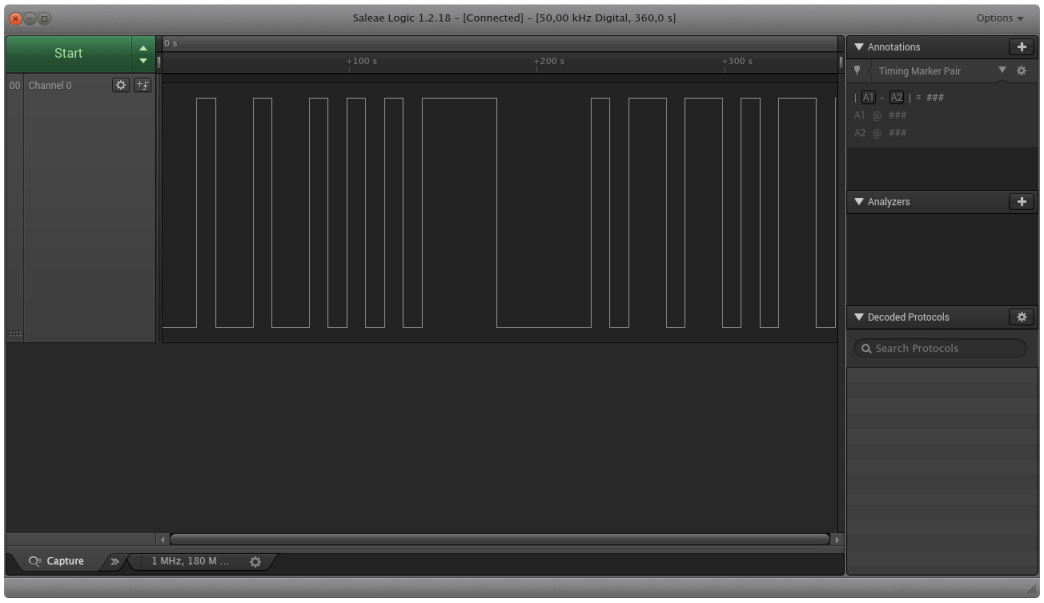

Figura 5.3: Resultado do teste de estímulos

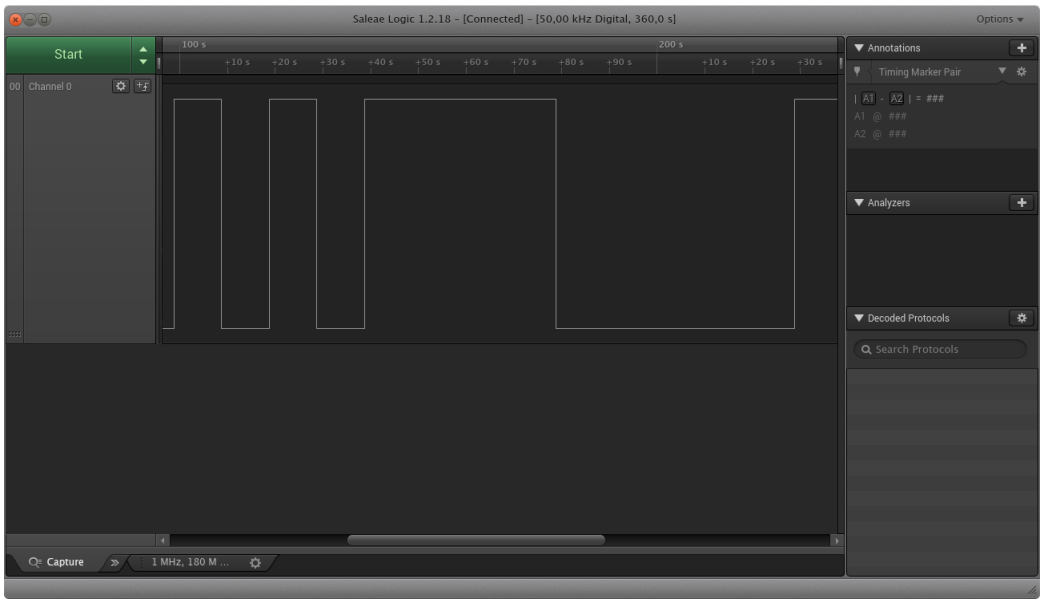

Figura 5.4: Observação do período de alteração de estado

Os resultados do experimento de obtenção de parâmetros do sistema é apresentado na figura 5.5, onde o 'Canal 0' representa a recepção dos dados, o 'Canal 4' representa a leitura do cabeçalho, o 'Canal 5' a execução do comando e o 'Canal 6' ao envio da resposta. Os sinais 'Canal 2' e 'Canal 3' representam a montagem do pacote e envio ao servidor separadamente, portanto equivalem ao 'Canal 6'. Estes sinais param durante a execução do comando pois são geradas na notificação de monitoramento que acontece após a execução do comando do computador. O 'Canal 1' é a representação do sinal de amostragem que é limitada a 200 micro-segundos, um valor maior que o desejado, mas dentro do esperado. A duração de cada tarefa é mostrada na tabela 5.1, na figura 5.5 é possível estimar apenas o valor total da latência, incluindo o envio da resposta, que é de aproximadamente 500 micro-segundos e esta

dentro do valor desejado.

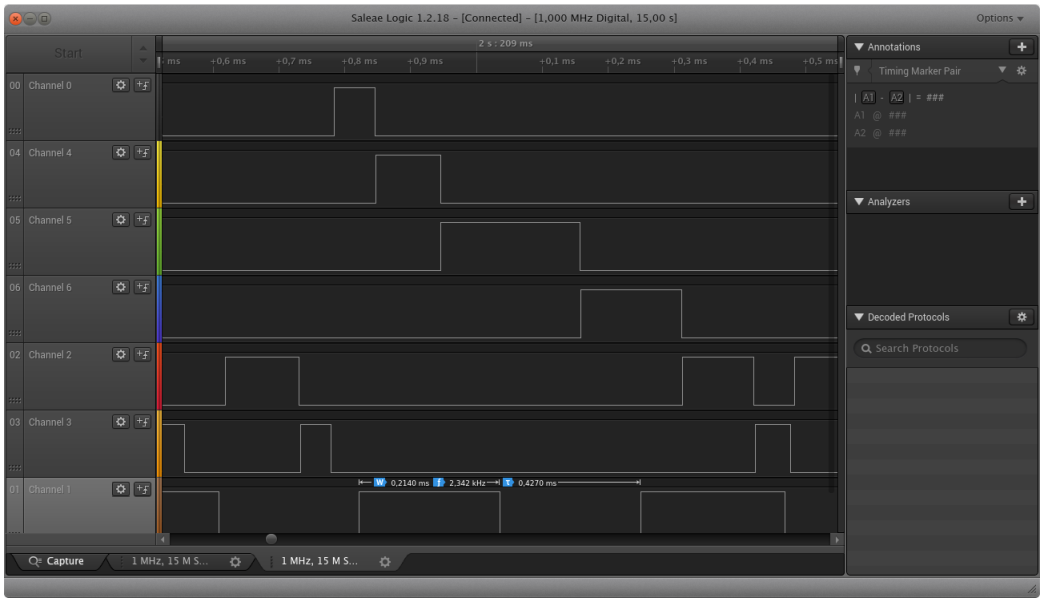

Figura 5.5: Sinais de visualização do tempo de execução do microcontrolador

Parâmetro	**Valor obtido**	**Valor estimado**
Atraso de recepção	60 μs	-
Atraso de processamento	100 μs	-
Atraso de execução	200 μs	-
Atraso de resposta	150 μs	-
Latência de envio	360 μs	< 1 ms
Latência total	510 μs	< 1 ms
Período mínimo de amostragem	200 μs	30 - 300 μs

Tabela 5.1: Parâmetros do sistema obtidos experimentalmente

Para a obtenção de parâmetros também foram testadas períodos de amostragem de 30, 60 e 100 micro segundos. Estes casos são mostrados nas figuras 5.6, 5.7 e 5.8. Onde podemos ver que a taxa de amostragem não correspondia a configurada ou não se mantinha constante durante o experimento, tornando o seu uso inaceitável e consequentemente diminuindo a taxa máxima de amostragem.

Os resultados obtidos pela realização dos testes foram satisfatórios e validaram o projeto, pois os objetivos foram alcançados. Foi possível a utilização do microcontrolador como dispositivo de geração e captura de sinais, assim foi possível observar e reconstruir o sinal de entrada. A geração de sinais permitiu a criação de formas de onda arbitrárias que, no contexto deste trabalho, é uma sequência pseudo-aleatória. As manipulações dos pinos foram realizadas exclusivamente pelo computador, permitindo assim a aplicação do projeto na arquitetura de laboratório remoto discutida no capítulo 1, um dos objetivos. A única interação entre usuário e a placa necessária foi a manipulação das chaves para simular os sinais da FPGA.

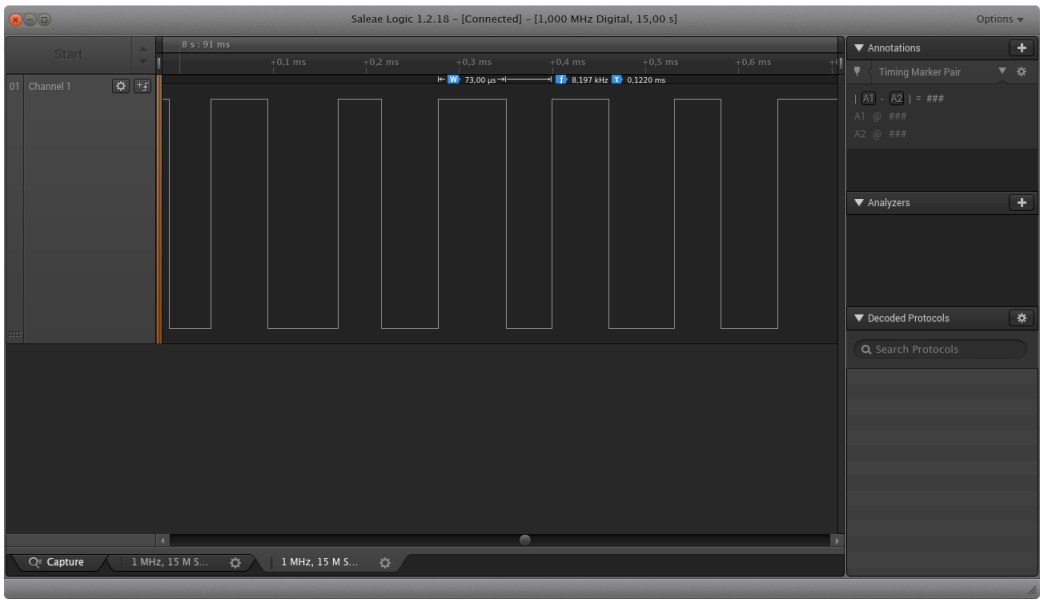

Figura 5.6: Amostragem configurada para 30 micro-segundos, a amostragem varia de 40 a 70 micro-segundos

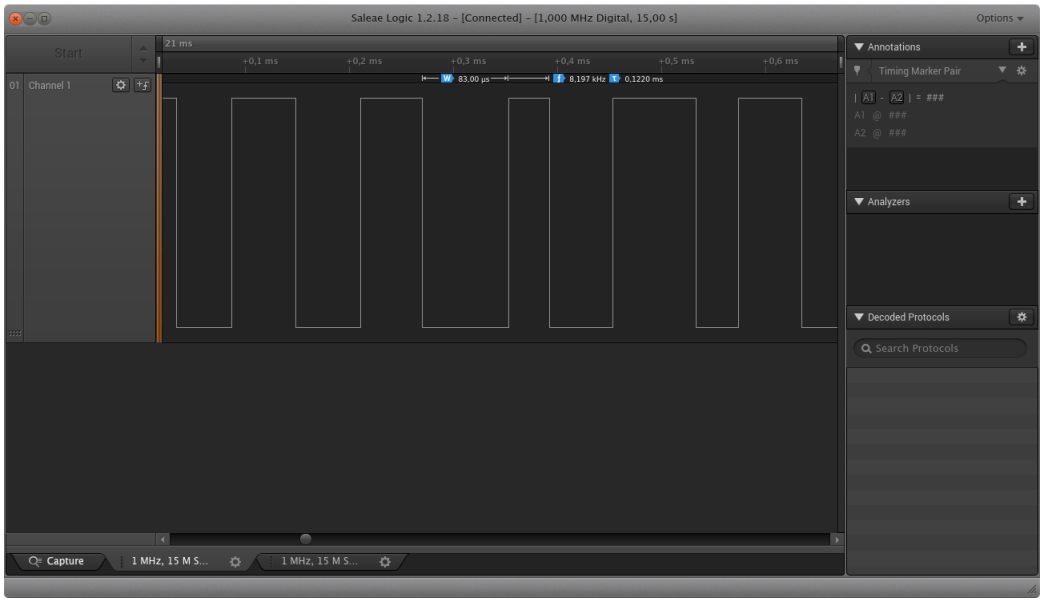

Figura 5.7: Amostragem configurada para 30 micro-segundos, a amostragem varia de 60 a 80 micro-segundos

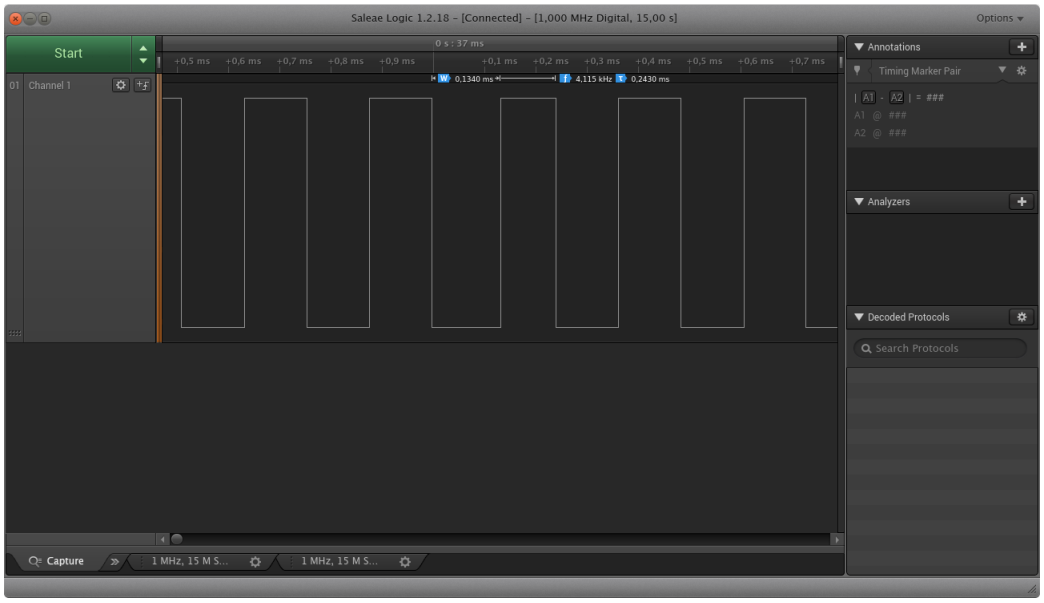

Figura 5.8: Amostragem configurada para 30 micro-segundos, a amostragem varia de 100 a 130 micro-segundos

Capítulo 6

Conclusão

Os testes e os resultados obtidos mostram que o trabalho atingiu os objetivos propostos. O desenvolvimento de um sistema de geração e captura de sinais e do protocolo de comunicação para controle do sistema pelo computador, permitindo sua manipulação de forma totalmente remota. O sistema é funcional, mas pode ser otimizado principalmente na utilização dos *timers* e dos periféricos do microcontrolador para a geração, captura e interpretação de sinais mais complexos, como comunicações seriais.

A utilização da interface USB no modo CDC (*Communication Device Class*) foi adequada para a realização do projeto. Permitiu a manipulação dos dados de forma simples, além de garantir a compatibilidade do protocolo desenvolvido no projeto para outros controladores além do MSP430 utilizado no desenvolvimento. A utilização de *bytes* como forma de representação dos dados, em vez de textos, também se mostrou bastante útil por tornar a manipulação dos dados pelo controlador mais simples e rápida, pois não há a necessidade de processamento complexo das mensagens recebidas antes da execução do comando.

A escolha do microcontrolador MSP430F5529 também foi adequada para o projeto, devido à disponibilidade de acesso, facilidade do desenvolvimento, documentação e exemplos disponibilizados pelo fabricante. Os recursos de *hardware* disponíveis no microcontrolador não tiveram grande influência na realização do projeto, pois sua utilização não se fez necessária. Assim qualquer controlador com uma interface serial e um conjunto de GPIOs, praticamente qualquer microcontrolador, também seria uma opção viável para a realização do projeto, assim o microcontrolador pode ser alterado no futuro caso seja necessário.

O protocolo desenvolvido no projeto também é satisfatório, pois a única informação dependente do microcontrolador é o identificador dos pinos, que utiliza a mesma notação dos microcontroladores MSP430, o que não é um problema, pois os pinos de outros controladores podem ser mapeados para esta notação, possibilitando que outros controladores implementem o mesmo protocolo de comunicação sem a necessidade de alteração em outras partes do sistema. O protocolo também é expansível para as necessidades de trabalhos futuros que busquem melhorar o *firmware* desenvolvido.

6.1 Trabalhos Futuros

Anteriormente foram mencionadas as limitações do *firmware* desenvolvido neste trabalho. Então uma proposta para trabalhos futuros é a melhoria do código desenvolvido visando eliminar tais limitações, sendo elas o uso do *timer* para a manipulação de no máximo 1 pino e a emulação de formas de onda complexas utilizando apenas GPIOs em vez do periférico adequado para geração deste sinal. Além deste trabalho futuro temos também a integração entre o servidor desenvolvido por [Araújo 2019] e o *firmware* desenvolvido neste trabalho; a implantação do sistema em um ambiente adequado para disponibilizar sua utilização pela internet e a integração do sistema com o protocolo LDAP como mencionado por [Araújo 2019].

Referências Bibliográficas

[Araújo 2019] Araújo, L. T. d. S. (2019). Sistema FPGA controlável remotamente. Monografia (Bacharel em Engenharia da computação), UnB (Universidade de Brasília), Brasília, Brasil.

[Gomes et al. 2009] Gomes, L., Gomes, L., Patricio, G., Patrício, G., Ferreira, R., Costa, A., and Costa, A. (2009). Remote experimentation for introductory digital logic course. In *2009 3rd IEEE International Conference on E-Learning in Industrial Electronics (ICELIE)*, pages 98–103. IEEE.

[Hashemian and Riddley 2007] Hashemian, R. and Riddley, J. (2007). FPGA e-Lab, a technique to remote access a laboratory to design and test. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, pages 139–140. IEEE.

[Indrusiak et al. 2007] Indrusiak, L. S., Glesner, M., and Reis, R. (2007). On the evolution of remote laboratories for prototyping digital electronic systems. *IEEE Transactions on Industrial Electronics*, 54(6):3069–3077.

[Morgan et al. 2012] Morgan, F., Cawley, S., and Newell, D. (2012). Remote FPGA lab for enhancing learning of digital systems. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 5(3):18.

[Rajasekhar et al. 2008] Rajasekhar, Y., Kritikos, W. V., Schmidt, A. G., and Sass, R. (2008). Teaching FPGA system design via a remote laboratory facility. In *2008 International Conference on Field Programmable Logic and Applications*, pages 687–690. IEEE.

[Soares et al. 2011] Soares, J., Lobo, J., and DEEC, F. (2011). A remote FPGA laboratory for digital design students. In *7th Portuguese Meeting on Reconfigurable Systems, REC*, volume 2011, pages 95–98.